A TLV Structure Semantic Constraints based Method for Reverse Engineering Protocol Packet Formats

Lian He, Qiao-yan Wen, Zhao Zhang
State Key Laboratory of Networking and Switching Technology
Beijing University of Posts and Telecommunications
Beijing, 100876, China
{hlpower, 108283, wqy}@bupt.edu.cn

ABSTRACT: *Mining unknown protocol packet formats is a very effective way to improve network security, especially in promoting the accuracy of network fuzz test. However, researches reverse engineering unknown protocol packets mostly depend on manual analysis, which is extremely time consuming and low efficiency. In this paper, we proposed a new method to infer the unknown protocol packet formats automatically. This method could infer the potential TLV fields and extract protocol format with low time consuming. First we define a threshold value for the sum of tag field types. Then we increase the value of a variable standing for the length of a tag filed until the type number of this tag fields reaches the threshold. After the tag filed is obtained, we can easily get the length field and the value field next to it. Run this process on the value field recursively, and we could finally get the whole structure of packet formats. In order to demonstrate the effectiveness, we applied our methods on the Get-Request packets of SNMP. As a result, almost 90% of the TLV structures of packets are extracted, at the same time, the field of Get-Request Id is also inferred successfully.*

Keywords: Reverse Engineering, TLV Structure, Packets Format, Protocol Analysis, Sequence Minning

Received: 11 November 2013, Revised 5 December 2013, Accepted 10 December 2013

1. Introduction

Extracting field formats from unknown protocols is a kind of reverse engineering, and the application-level protocol reversing is especially useful for network security. For example, in network robust testing, we can reduce the testing cases and broaden the coverage of fuzz test [1]. Whilein the intrusion-prevention system [2] [3], protocol analyzer is an important mechanisms for intrusion detection and firewall systems to perform deep packet inspection. However, most researches are done by manual analysis, which is a time-consuming and error-prone job. For example, it took an open-source SAMBA[4] project 12 years to manually reverse the Microsoft SMB protocol. Some methods of reversing engineering unknown protocol formats are hard to operate, such us Dicoverer [5]. This method is based on distinguish printable character from unprintable character, but this assumption is not applied for most protocols. Other open source projects are suspended as the protocols used for communication change frequently. Therefore, how to obtain packet formats and semantic information of fields is still an open problem.

There are two branches in protocol reverse engineering to extract packet formats: The first one focus on network traffic traces, which analyses packets from both sides of the conversation [5] [6] directly. And the other one analyses the execution programs [7] by tainted dynamic technology, which could extract the packet format more accurate than the former one, as we can obtain how a packet is constructed and parsed. However, in most cases, we could not have the privilege of getting access to the source code.

In this paper, we propose a method based on the network traffic traces to reverse engineer unknown TLV packets. We use this method to get the structure of the message format and some semantic information. The contributions of our research are as follows:

• Apply a highly efficient way to filter network traffic to get representative data samples.

• Extract the packet information from the data samples by forward analysing the structure of TLV packets iteratively, with a lower time consuming method compare to the bioinformatics algorithm [8].

• Propose a method to get the best result of hierarchical structure of unknown protocols.

2. Application Scenario

There are many kinds of patterns applied in protocol specifications, such as, the key-value pattern, the division pattern and the TLV pattern [9]. In the key-value pattern, every value field corresponds to a key field, while the byte's number of key and value fields is constant. The division pattern has some symbol for dividing packets, so we can get structure of packets by recognizing division symbols. And the TLV pattern uses the tag-length-value structure to describe the packet formats. The TLV pattern supports variable length fields compared to the key-value pattern, which is more robust than the division pattern. So the TLV pattern is used much more widely.

In this paper, we focus on how to reverse a protocol, which has a high possibility in TLV pattern. There are three restrains in the TLV pattern:

• The order of the sequence must be: the *T* field appears on the head, the *L* field is next to *T*, and the *V* field is at the end.

• *L* field and *V* field must satisfy: Value (*L*) = Length (*V*) Value (*L*) means the value of the *L* field, and Length (*V*) stands for the number of bytes of the *V* field.

• The number of tag types is under a threshold value.

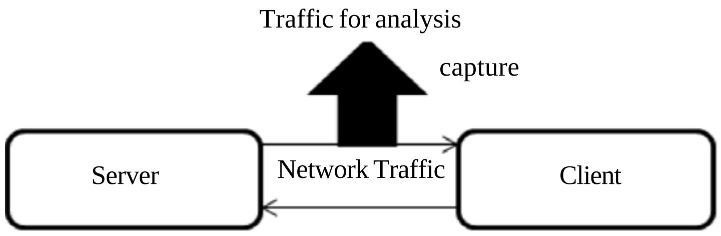

Figure 1. Application Scenario

The application scenario of our approach is illustrated in Figure 1 There are two communication entities, the server terminal and the client terminal. We capture the network traffic by WireShark [10] or TcpDump [11] and store it into some temporary files. These files are our input data samples. When collecting network traffic traces, we have to obey some rules to make the trace files more representative:

• Experiments should be operated for several times, and at each time the input data sample should be obtained under different situations, such as changing the IP address and querying information.

• Packets must be sent from the same protocol implementation.

• Packets must be captured in increasing time to make the request/response id progressive increase

• The length of packets should be different to make the *L* field different from each other, and the length must be less than 255 bytes.

3. Algorithm Design

3.1 Condition

This algorithm can be designed when taking the following conditions into consideration.

1) The types of *T* field are limited.

2) The *L* field occurs before the *V* field, so that it's convenient to deal with the mutable-length field.

3) The request/response id is progressive increasing.

4) As the packet length is less than 255 bytes, we can assume the *L* field is only one byte.

3.2 Design

Algorithm: **getTreeStruct**

Input: A group of packets with the same format: *P*1, *P*2,… , *Pm*, Number of *T* filed 's number *X*.

Output: Field tree with inferred VTL fields and corresponding referred fields. Describe: We use this function to get the tree structure of input parameters.

```
Do
    For i = 1..length (P)
        If (types (P1[0..i], P2 [0..i],…Pm [0..i]) < X)
            if (isIdProgressiveIncrese (P1[i], P2[i],…Pm [i])
                continue;
            else if (i = =1)
             return P;
       else
            [TLV1… TLVm] = DevidePacket (P1..Pm, i)
         TreeStruct1 = getTreeStruct (V1,V2,..Vm)
          Treestruct2 = getTreeStruct (P1', P2',..Pm');
return constructTree (TreeStruct 1,TreeStruct 2)
```

Figure 2. Pseudo-code of this get TLV structure algorithm

types () gets the total number of different values given as the input parameter. **isIdProgressiveIncrese** () is used to check whether the values of the input are increasing monotonously. While function **constructTree** () combines two tree into one tree structure by adding a root node for them.

The TLV_i fields stand for the first level of the *TLV* structure of bytes stream given as *i*th input parameters. Then the *Vi* field is the *V* part of TLV_i. And the *Pi*' field is the remaining part of *i*th input parameters after we inferred the TLV_i fields.

As assumption in the condition of section 3, the number of the *T* field type is limited. We check every packet forward with length of *i* to see how many different substrings there are, and set these substrings as tag fields.

We add one to variable *i* each time until the number of tags types grows more than the threshold whose value is *X*. Then the field on the *i*th position may have three states:

First, request/response id. If the values of this field are progressive increasing. We just treat it as the *T* field, and keep on increasing *i*.

Second, the minimum unit of the *V* field. If the value of variable *i* equals to one and the number of the *T* types is already over the threshold, we regard this packet as an entirety unit and we don't need to mine recursively on this field.

Third, the *L* field. As most packets we collected have different lengths, this field has a high possibility to be the *L* field if it does not satisfy the other two states.

After we figure out the *L* field, we can obtain the *TLV* structure of this level. Then we run this process recursively on the *V* field to get the whole field tree. Figure 3 shows the first level of the recursion, *P'* means the remaining field of *P* after the first recursion, Figure 4 shows the second recursion, which is operated on *V* and *P'* field.

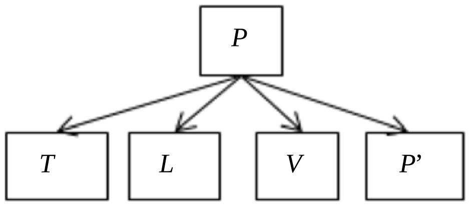

Figure 3. First level of recursion

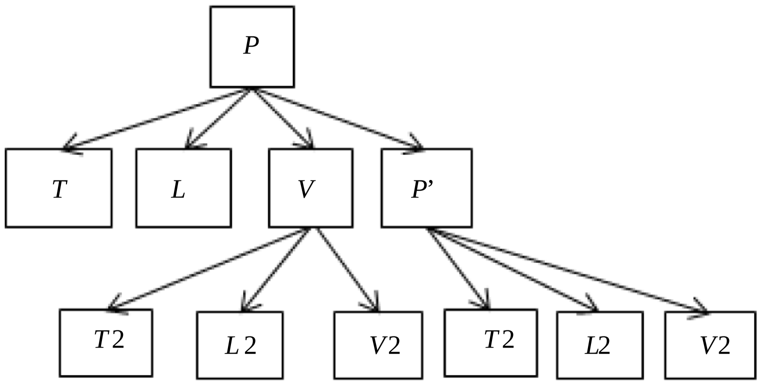

Figure 4. Second level of recursion

We can get different tree structures of packets by changing the input parameter X, We select the most stable structure among them, that is to say this structure changes least when using other data samples as input files.

4. Experiment And Analysis

We applied two indicators to evaluate the reliability of the algorithm: the rate of the TLV structures that are found, and the error rate of these structures.

We applied our approach to the data of SNMP V1 which contains many TLV structures, so that we can test our algorithm expediently. We treated SNMP V1 as an unknown protocol, and then analyzed its packets by our algorithm., We can evaluate the algorithm by comparing the analysis result with the specifications. as the specifications of SNMP V1 can be easily obtained on the Internet.

First, we captured the network traffic and save it into a pcap file. Then, filtered this file according to the standard proposed in section 2, and get the input data samples. We extract their formats by our algorithm for several times. And for each time, we change the threshold value. Finally, compare the results to get the most stable one.

Figure 3 shows an example in which the threshold value is set to 2. The format and the semantic information we extracted is as follows:

• *T*: The tag field, whose value changes in a limited range. In this experiment, it can be the community field, the version field or some separator fields.

• *L*: The length field of this layer.

• *V*: The value field, which contains the information to transport.

• *Index*: The counter field whose value increases monotonously.In this experiment, it is the request-id field.

• *T* (terminate) : The indeterminate tag field. We treat it as the type field because it changes under the threshold value. The field is too long, and it may contain nested structures inside. However we can’t get the information from the input data samples. By improving the representation of the data samples, we can solve this problem.

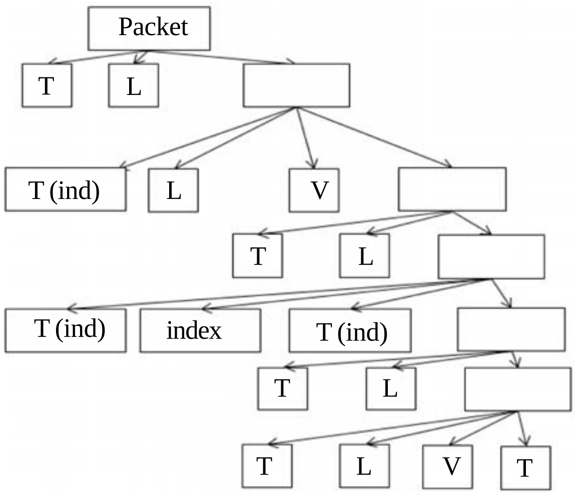

Figure 5. Packet structure we mined

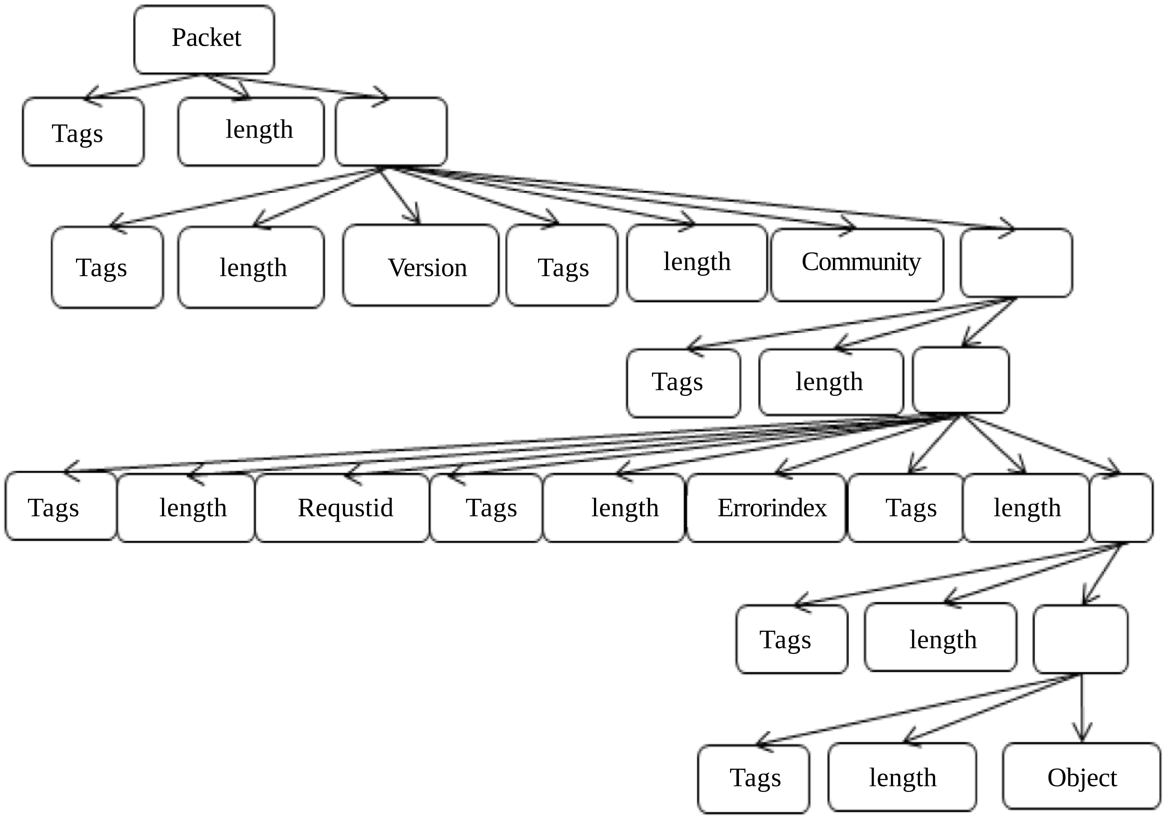

Figure 6. SNMP v1 Get-Request packet

Figure 5 shows the format specification of SNMP V1. Compare our result with the format specifications, we can get the following conclusions:

• There are 9 TLV structures in the SNMP specification in total. In the result, we found 5 of them and took 3 into the indeterminate fields. In other words, with enough representative data samples, we can make the found rate of TLV structures approximate to 89 percent.

• The error found rate is 0 in this experiment.

• The hierarchy structure we extracted nearly approximates to the real packet.

• We got the id field, but couldn't infer the semantic of other type fields.

Although we did not get the whole format of the given protocol, these features obtained can obviously improve the fuzz test and intrusion detection systems. By get more typical data samples, we can get more information of the format.

5. Complexity Analysis

The computational complexity analysis is done for TLV structure semantic constraints based method for reverse engineering protocol packet formats. In order to start the complexity analysis, we have to declare some variables:

• ***N*****:** The number of packets in the data samples.

• ***M*****:** The average length of packets in the data samples.

The complexity includes space complexity and time complexity. We increase the length of T by one each time, and check the packets of every data sample. If the length is right, we don't roll back the bytes, which have already been checked. It costs $O(N)$ for each check, so the time complexity of the whole process is $O(MN)$. It is obviously faster than some automatic reversing methods, such as methods based on MSA.

In order to save check result of each step, we need the space complexity of $O(N)$. The construction of the tree structures of the packets need $O(M)$, so the space complexity is $O(M + N)$.

6. Conclusion

In this paper, we make some improvement to previous researches. We can use this algorithm to get some protocol structures and even some semantic information. The experimental results demonstrate our approach is available in network security, especially for protocol fuzz test and intrusion detection systems.

7. Acknowledgment

This work is supported by NSFC (Grant Nos. 60873191, 60903152, 61003286, 60821001) and the Fundamental Research Funds for the Central Universities (Grant Nos. BUPT2011YB01, BUPT2011RC0505).

References

[1] Li Weiming, Zhang Aifang, Liu Jiancai, et al. (2011). An automatic network protocol fuzz testing and vulnerability discovering method [J]. *Chinese Journal of Computers*, 34 (2) 242-255.

[2] Paxson, V. (1999). Bro: A system for detecting network intruders in real-time, *Computer Network*, 31 (23-24) 2435-2463.

[3] Dreger, H., Feldmann, A., Mai, M. et al. (2006). Dynamic application-layer protocol analysis for network intrusion detection [C] *In*: Proc of the 15th USENIX Security Symposium. p. 257-272.

[4] Tridgell, A. How samba was written [OL]. http://samba.org/ftp/tridge/misc/french_cafe.txt.

[5] Weidong Cui, Jayanthkumar Kannan Helen J.Wang. Discoverer: Automatic Protocol Reverse Engineering from Network Tracs .

[6] BEDDOE, M. Protocol information project [EB/OL]. (2004-10-05) http://www.4tphi.net/~awalters/PI/PI.html.

[7] Zhiqiang Lin, Xiangyu Zhang, Dongyan Xu. Reverse Engineering Input Syntactic Structure from Program Execution and Its Application, IEE Transaction on Software Engineering, 36 (5) 688-704

[8] Beddoe, M. A. (2004). Network protocol analysis using bioinformatics algorithms, http://www.baselineresearch. net/PI/.

[9] Hang Liu, Dan Zhang Communications (ICC), IEEE International Conference 5822-2827 A TLV-Structured Data Naming Scheme for Content-Oriented Networking.

[10] Wireshark [OL]: The world’s Most Popular Network Protocol Analyzer. http://www. Wireshark.org/

[11] TCPDUMP & LiBPCAP[OL]. http://www.tcpdump.org.